Unix Survival Guide

For a mixture of historical and practical reasons, much of the bioinformatics software discussed in this series runs on Linux, MacOS X, Solaris, or one of the many other Unix variants. This appendix provides the minimum information needed to survive in a Unix environment.

LOGGING IN AND OUT

Unix dates from the time when computers were very expensive, necessitating that multiple users share the same computer hardware. For this reason, a session on a Unix system begins with a login prompt. You provide the system with a username and password in order to gain access to the system's resources. If your Unix system is managed by a system administrator from your institution's Information Technology (IT) department, the username and password will have been assigned to you. If you have installed Unix yourself, you will have been prompted for a username-password pair at the time of installation.

There are two common login scenarios. In the first, you are sitting in front of the Unix computer itself and are using its monitor and keyboard directly (a situation sentimentally called "logged in at the console"). In the second, you use a conventional Windows or Macintosh desktop machine to connect via the network to a Unix server located at some remote location.

Logging in at the Console

In the first scenario, you will be presented with a login window. A typical login window is shown in Figure A.1C.1, but because of the great variability in Unix distributions yours will almost certainly look a bit different; however, all login windows have a field for Username and another for Password. Type yours in and press the appropriate button (Go! in the example shown in the figure).

If the username and password are recognized, the system will log you in and display a graphical desktop (Fig. A.1C.2). Like the login prompt, Unix systems vary widely in the appearance and behavior of the desktop. Some, such as the KDE desktop shown in the Figure A.1C.2, do a good job of reproducing the familiar experience of a windows or Macintosh desktop. Others are frustratingly alien. All require some getting used to. Popular Unix desktop systems that you may encounter include the aforementioned K Desktop Environment (KDE), Gnome, and the Common Desktop Environment (CDE).

It would be impractical to give a full tutorial on navigating all the Unix desktop variants here, but a few hints will help you get started. First, many desktops make extensive use of the right mouse button. If in doubt about what to do next, pressing the right mouse button on the desktop, within a window, or in a window title bar often brings up a menu of possible commands. Some desktops also make use of the middle mouse button, which is a standard feature of Unix workstations but is not found on many PC mice. To emulate the middle mouse button, try pressing the left and right buttons simultaneously. Finally, most desktops have a built-in tutorial and help system which can usually be activated without too much flailing.

To log out of the desktop, look for menu items with names like "Log out," buttons with the power on/off icon found on some electronic appliances, or icons that show a moon and stars.

Contributed by Lincoln D. Stein
Current Protocols in Bioinformatics (2006) A.1C.1-A.1C.24

Figure A.1C.1 A typical login window for “logging in at the console.”

Figure A.1C.2 K Desktop Environment (KDE) at point of successful login.

Logging in Remotely

If the Unix system you wish to access is located remotely, you will use one of several remote access programs to log into it from your desktop machine. These programs range from extremely bare-bones terminal emulators that provide you with a 24-line by 80-character text-only window to sophisticated graphical emulators that will display the Unix graphical desktop on your PC or Macintosh.

Which terminal emulation program you use depends on the capabilities of your desktop machine, the configuration of your local area network, and what software is installed on the Unix machine. Typically, your system administrator or IT department will tell you what remote access software to use. Common remote login packages are listed below (see Internet Resources).

More information on using the graphical remote access protocols based on VNC and the X windows systems are given in *APPENDIX 1D*. Here we will assume that you will be logging in using a text-only terminal emulator.

Logging into a remote system from a PC

If you are on a Microsoft Windows 95 or higher system, a simple terminal emulator is already installed on your system; however, it is a bit hidden. Select Run Command... from the Start menu, and when prompted type in `telnet`. This starts up the Telnet program which displays a plain white window and simple menu bar. From the Telnet window's Connect menu, select Remote System... to bring up a dialogue box that prompts you for the name of the host with which to connect and the connection settings you wish to use. By and large, the default connection settings will work, so don't change them. Just enter the name of the Unix machine to which you wish to connect (using its dotted internet name or address) and press Connect.

Telnet will now attempt to connect to the indicated machine. If successful, the terminal window will display a login prompt (Fig. A.1C.3). Type your login name and password, pressing Enter each time. If you successfully log in, the remote host will print a greeting, a status message, and possibly a pithy quote of the day as shown in the figure. The remote host will start the command-line shell, and print an input prompt, which is shown in Figure A.1C.3 as the cryptic series of characters (~) 51%.

Since Unix is a multiuser operating system, you can log into the same system multiple times. Simply repeat the login procedure described above as many times as you wish.

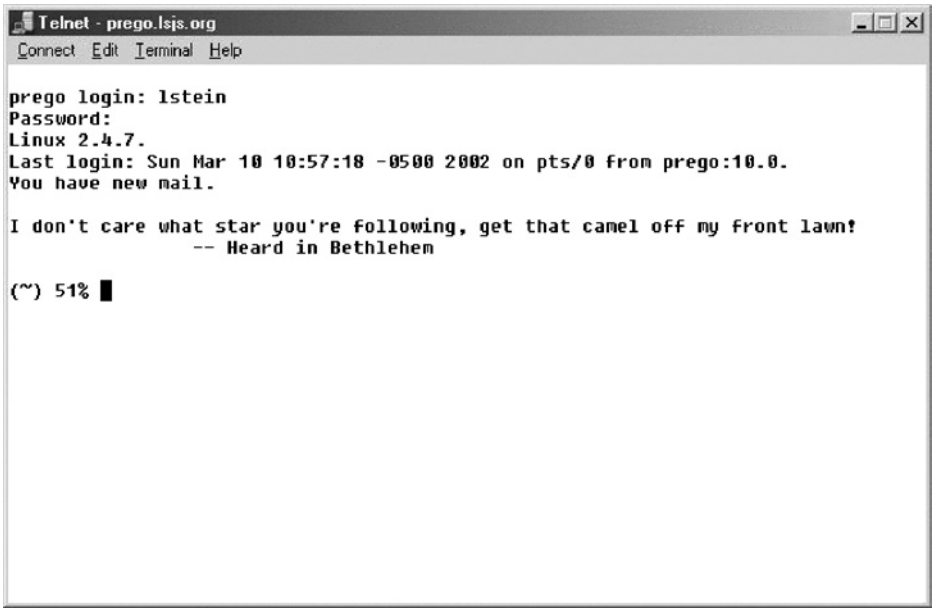

Figure A.1C.3 Successful remote login using Telnet.

Logging into a remote system from a Unix system

If you are using a Unix system and wish to log into another system remotely, open up the command line window. Depending on how your system is configured, this may be called a "shell window," "command window," or "terminal window." Now, use one of the terminal emulators Telnet or SSH to connect to the remote machine. Telnet uses an older communication protocol that sends all text across the network in unencrypted form. Ssh uses a newer protocol that encrypts all outgoing and incoming communication. Because of security concerns, we highly recommend SSH if it is available, but using it requires that it be supported by both the local and the remote machines, which is usually, but not always, the case.

To connect to the remote machine named `host.example.com` using Telnet, type `telnet host.example.com` at the command line and press return. As described earlier, Telnet will attempt to contact the named machine, and, if successful, will prompt you for your username and password. SSH works similarly; imply type `ssh host.example.com`.

Logging into a remote system from a Macintosh

Recent versions of the Macintosh run Mac OSX. Mac OSX is itself a Unix system, and most, if not all of the bioinformatics tools described in *Current Protocols in Bioinformatics* will run on a Macintosh under this platform. However, some of the tools that have graphical user interfaces will require installation of X Window server software. This is described in *APPENDIX 1D*, X Window Survival Guide.

In order to log into a remote Unix machine (e.g., Linux or even another Macintosh running under OSX), you will use the command line tool that comes standard with Mac OSX. Using the Finder, look for the Terminal application in the Applications area in the Utilities folder. Double-click the Terminal to launch it, then proceed as described for the Unix login. SSH is standard in Mac OSX, and we highly recommend that you use that application if the remote host supports it.

Logging out

To quit a terminal emulator session, you may either close its window or type `logout` at the command line prompt. You can also quit the emulator application entirely, but this will have the effect of closing all open sessions.

USING THE COMMAND SHELL

Despite the graphical desktop environments now becoming prevalent, Unix is still very much a command-line oriented system. You issue instructions to the system by typing cryptic commands in a terminal window, and the output of programs are displayed as text inside the same window. Most bioinformatics packages are command-line oriented, and even for those few that use windows, menus, and mouse clicks, you will still have to install and possibly invoke them from the command line.

The Unix program that accepts and processes commands is called the "shell." It is a simple program that prints out a command-line prompt, waits for you to type a command and press the Enter key, and then runs the command. After the command is complete, the shell again prints the command line prompt, awaiting further instructions.

If you have logged into the system using a terminal emulator, you are already running a shell. Otherwise, if you are using a graphical desktop, you will need to launch a terminal emulator within the desktop environment in order to interact with the shell. To do this, look for a menu command called Shell, Terminal, Console Xterm, or some variant of

```
(~/docs)  72%
(docs)  495>
bash>
lstein@prego.lsjs.org>
lstein@pesto:~>
72%
$
```

Figure A.1C.4 Some shell command-line prompts.

the above. Icons that launch terminal emulators take the form of stylized shells or little desktop PCs. Running one of the emulators creates a terminal window similar to those used by the Windows and Macintosh emulator programs. One advantage provided by the desktop environment terminal emulators is that you can resize them at will. You can also launch multiple emulators, and each one will run a different shell session.

Regardless of whether you have logged in graphically or remotely, the terminal emulator will be displaying a command-line prompt. The exact appearance of this prompt depends on the variant of Unix you are using, which shell program (there are several), and how the system has been configured. A few common command-line prompts are shown in Figure A.1C.4. Prompts typically contain a short amount of status information (e.g., the time of day, your login name, the hostname, or the number of commands you have typed) followed by one of the characters "%", ">", or "$".

Working at the command line will be a foreign experience to many readers. Although it will never be completely painless, a few features do make working at the command line easier. First, most command-line shells offer in-place editing. You can use the left and right cursor keys to move the text insertion point back and forth on the command line in order to insert and delete characters. The backspace key will delete characters to the left of the insertion point, and the delete key, or sometimes Control-D, will delete characters to the right of the insertion point.

If you find yourself repeating many commands with minor variations, the up (↑) and down (↓) cursor keys will activate the shell's "command history" feature. Pressing the up-cursor key will insert the last-issued command at the prompt. Pressing ↑ again will fetch the command previous to that, and so forth. You can press Enter to reissue the command, or use the cursor keys to edit the command prior to issuing it again.

Most shells also offer a "command completion" feature. With this feature, you can type the first few letters of a command or file name and then press the Tab key. The shell will complete the command for you, or, if what you typed was ambiguous, display a number of alternatives from which to make a selection

Command Syntax

Unix commands are case-sensitive, meaning that the commands `mkdir` and `Mkdir` are not the same. The first command will create a new directory. The second is not recognized on typical Unix systems and will result in a `Command not found` warning. Unix commands typically take arguments that are separated from the command name by one or more spaces called "whitespace." As a concrete example, the `mkdir` command takes a series of arguments giving the names of the directories to create. This command will create three directories named "docs," "toy," and "experiments":

```
(~) 51% mkdir docs toy experiments
```

To pass an argument that contains whitespace, surround it with double or single quotes. In contrast to the previous example, this one will create two directories, one named “docs” and one named “toy experiments”:

```
(~) 51% mkdir docs ‘‘toy experiments’’
```

Options

Many Unix commands accept “options” which modify their behavior. Depending on the command, its options may be single-letter codes preceded by a hyphen, as in `-v`, or fully spelled-out words preceded by two hyphens, as in `--verbose`. Options come after the command name and before any arguments. For example, to have the `mkdir` command print out what it is doing, use the `--verbose` option:

```
(~) 51% mkdir --verbose docs toy experiments
  mkdir: created directory ‘docs’
  mkdir: created directory ‘toy’
  mkdir: created directory ‘experiments’
```

Getting Information on Commands

When given the `-h` or `--help` options, most commands will print out a brief usage summary. Try `-h` first, and if that doesn’t work try `--help` as shown in Figure A.1C.5.

Manual command

For more detailed help the `man` (manual) command is extremely useful. Invoke it with the name of the command you wish help on (e.g., `man mkdir`). This will display a page of detailed information on how to use the command. If you don’t know the name of the command for which you are looking, try the `apropos` command (e.g., “apropos directory”) to generate a list of commands that might have something to do with the function for which you’re looking.

The `man` command may use a “pager” to display a manual page that is longer than will fit comfortably into a terminal window. The pager is very simple. It displays a single

A

```
(~) 51% mkdir -h
  mkdir: invalid option -- h
```

B

```
(~) 52% mkdir --help
  Usage: mkdir [OPTION] DIRECTORY...
Create the DIRECTORY(ies), if they do not already exist.
-m, --mode=MODE   set permission mode (as in chmod), not rwxrwxrwx - umask
-p, --parents     no error if existing, make parent directories as needed
-v, --verbose     print a message for each created directory
--help            display this help and exit
--version         output version information and exit
Report bugs to <bug-fileutils@gnu.org>.
```

Figure A.1C.5 To get help on the use of the `mkdir` command, the (**A**) `-h` option is used (unsuccessfully), followed by the longer (**B**) `—help` (successful).

page for your perusal. When you are ready for the next page, hit the Space Bar. When you are done reading, press "q" to quit. Some systems have more sophisticated pagers that will allow you to page up and down a line at a time using the cursor keys, or a page at a time using the Page Up and Page Down keys. Experiment a bit to see if your system supports this.

Suspending and Killing Commands

At some point while working with the command shell you will issue a command that either produces large amounts of output, takes a long time to run, or does something unexpected. In this case, you can interrupt a command in either of two ways.

To interrupt a command before it has finished running press Control-C. This means to press the Control key (marked Ctrl on most PC keyboards) and simultaneously press the (lowercase) "c" key. In most cases this will interrupt the command and return you to the command prompt. In rare cases you may need a more emphatic type of interruption. Try Control-\ (i.e., backslash while holding down the Control key).

To temporarily suspend a command without killing it entirely, press Control-Z. This will put the command into suspended animation and return you to the command prompt. To resume the command, type `fg` (foreground). You can suspend and resume a command as many times as you like.

All the Unix commands we have seen so far are short lived. For example, the `mkdir` command does its work and returns almost instantly. However, other commands are long lived. This is particularly true of commands that launch graphical programs such as Web browsers or text editors. In such cases you will not be able to use the command line until the program has finished executing and the command-line prompt has reappeared.

To avoid losing the use of the command line, you can place an ampersand "&" after the name of a program that will take a long time to execute. This will place the program in the "background" and return you to the command-line prompt immediately. For example, the command `netscape &` will launch the Netscape Web browser in the background. The Netscape window will appear, and you will be returned to the command-line prompt in the terminal window.

If you forget to add the ampersand and lose your command line, you can temporarily suspend the running program by typing Control-Z in the terminal window. The command-line prompt will reappear. Type `bg` (background), and the suspended program will be restarted in background mode.

MANAGING FILES AND DIRECTORIES

Like other operating systems, a fundamental part of Unix is its support for files and directories. A file can contain text, computer code, word processing data, images or sounds, or any other data. A directory, equivalent to the Macintosh and Windows "folder," contains files and/or other directories.

If you are logging in via a terminal emulator, you will have to learn to work with files via the command line. If you have a graphical login, chances are that the desktop environment provides a file browser. With the browser, you can view the contents of directories, peek into files, create new directories, move existing files and directories around, and so forth. Even so, you will need to learn the basic shell commands for manipulating files and directories.

Unix has the concept of the "current working directory," the default directory that the various file manipulation commands operate on if not otherwise specified. When you first log in, the current working directory is set to your "home directory," a directory to which you have full access and where you will normally store your personal files and other data.

List Command

To see the contents of your home directory, issue the `ls` (list) command (Fig. A.1C.6).

Fancy option

The `ls` command shows a formatted list of files and directories, but doesn't provide any indication about which is which. For a more informative display, use the `-F` (fancy) option (Fig. A.1C.7). The `ls -F` command shows a marked-up version of the directory listing. Directories end in the slash character "/", executable files (those that contain computer code) end in an asterisk "*", symbolic links (a type of alias or shortcut) end in the "at" character "@" while regular files have no special character at the end.

Some of the files shown in Figure A.1C.7 are text files. An example is `INBOX`, which contains a list of recent E-mail messages to the author. Others contain image data such as `chloroplast.png` and `plastid.png`, which are both images of genomic annotations of the rice chloroplast. Unix distinguishes file types by using distinctive file name extensions. For example, `.png` is used for a file that contains portable network graphics image data. Unlike some systems, where file extensions are limited to three characters, Unix extensions can be of any length.

Long version option

Another useful variant of `ls` is the long version, invoked with `ls -lF`. This adds detailed information to the listing. This form will tell you how large the file or directory is, which user owns it, and what its access permissions are (Fig. A.1C.8).

The first column of the long listing indicates the file permissions and its interpretation is beyond the scope of this appendix; however, it is handy to know that the `d` that sometimes appears at the beginning of the column indicates that the corresponding item is a directory.

```
(~) 66% ls
 Desktop   autosave          docs   jcod   mutella      projects
 INBOX     bin               etc    lib    nsmail       public_html
 Mail      build             foo    .pl    lstein       pcod src
 News      chloroplast.png   games  man    plastid.png  tmp
```

Figure A.1C.6 Example output of the `ls` (list) command.

```
(~) 67% ls -F
 Desktop/  autosave/        docs/     jcod/     mutella/     projects/
 INBOX     bin/             etc/      lib/      nsmail/      public_html/
 Mail@     build/           foo.pl*   lstein/   pcod/        src/
 News/     chloroplast.png  games/    man/      plastid.png  tmp/
```

Figure A.1C.7 Example output of the `ls` (list) command with `-F` (fancy) option.

```
(~) 68% ls -1F
total 1684
drwx------    5  lstein   lstein       224  Mar 10  11:05 Desktop/
-rw-r--r--    1  lstein   lstein   1639715  Mar  8  06:36 INBOX
lrwxrwxrwx    1  lstein   lstein        11  Oct  6  23:35 Mail -> lstein/Mail/
drwxr-xr-x   15  lstein   lstein      2944  Mar  8  07:08 News/
drwxr-xr-x    2  lstein   lstein        48  Feb 10  19:06 autosave/
drwxr-xr-x    2  lstein   lstein      1248  Mar  6  07:53 bin/
drwxr-xr-x   52  lstein   lstein      1808  Feb 28  04:57 build/
-rw-r--r--    1  lstein   lstein     18647  Mar  8  06:34 chloroplast.png
drwxr-xr-x   63  lstein   lstein      2544  Feb 15  04:46 docs/
drwxr-xr-x    3  lstein   lstein       120  Oct 25  1997  etc/
-rwxr-xr-x    1  lstein   lstein       287  Feb  2  19:47 foo.pl*
drwxr-xr-x   17  lstein   lstein      2128  Jul 19  2001  games/
drwxr-xr-x    5  lstein   lstein       256  Feb 17  2001  jcod/
drwxr-xr-x    7  lstein   lstein       224  Oct 19  10:11 lib/
drwxr-xr-x    7  lstein   lstein       176  Oct  6  1999  lstein/
drwxr-xr-x    3  lstein   lstein        72  Oct 19  10:10 man/
drwx------    3  lstein   lstein      1408  Mar  9  08:15 mutella/
drwx------    3  lstein   lstein       472  May 11  2001  nsmail/
drwxr-xr-x   33  lstein   lstein      2080  Aug 31  2000  pcod/
-rw-r--r--    1  lstein   lstein     26988  Mar  8  06:34 plastid.png
drwxr-xr-x   76  lstein   lstein      2960  Feb 27  04:37 projects/
drwxr-xr-x    5  lstein   lstein      1016  Dec 24  11:27 public_html/
drwxrwxr-x    3  lstein   lstein      2968  Feb 28  04:50 src/
drwxrwxr-x   10  lstein   lstein       576  Mar 10  14:19 tmp/
```

Figure A.1C.8 Example output of the `ls` (list) command with `-lF` (long version) option.

All option

By convention, Unix uses files whose names begin with a period (.) to hold software configuration information. Since there are many of these in your home directory, the `ls` command skips over these hidden files by default. To force `ls` to show all files, including those that are ordinarily hidden, use the `-a` option:

```
(~) 68% ls -a
.ptksh_history                .qtella
.DCOPserver_pesto@            .qtella.hosts
.FVWM2-errors                 .registry*
.FVWM95-errors                .rhmapper
.ICEauthority                 .rhosts
.MCOP-random-seed             .rnd
...
```

Directory Paths

To view the contents of a directory, you have several options. One is to use the directory name (e.g., `docs`) as the argument of the `ls` command (Fig. A.1C.9). The contents of the `docs` directory is mostly other directories. We can peek down even further by providing `ls` with a directory "path." A path is simply a list of directories separated by slashes (Fig. A.1C.10).

If Unix paths remind you of Web URLs in any way, that isn't a total coincidence. The Web was originally built on top of Unix.

```
(~) 68% ls -F docs
GKB/            cp_bioinformatics/  mod_perl_book/        networking_tutorial/
WebTechniques/  das/                mod_perl_book.tar.gz  perl_networking_book/
bind/           dogs/               nature_genetics/      talks/
```

Figure A.1C.9 Viewing the contents of the `docs` directory using the `ls` command.

A

```
(~) 69% ls -F docs/talks
 apache_security/  networking1/  networking2/
```

B

```
(~) 70% ls -F docs/talks/networking1
basic1.html   examples/          index.html                    pipes.html
easy.html     filehandles.html   objectfh.html                 sockets.html
easy2.html    i/                 perl_network_programming.jpg  talk.css
```

Figure A.1C.10 Viewing the contents of the (**A**) `talks` subdirectory of `docs` (path: `docs/talks`) and the (**B**) `networking1` subdirectory of `talks` (path: `docs/talks/networking1`) using the `ls` command. Also see Figure A.1C.8.

```
(~) 71% cd docs

(~/docs) 72% ls -F
GKB/            cp_bioinformatics/  mod_perl_book/        networking_tutorial/
WebTechniques/  das/                mod_perl_book.tar.gz  perl_networking_book/
bind/           dogs/               nature_genetics/      talks/
```

Figure A.1C.11 Viewing the contents of the `docs` directory by first changing the current working directory with the `cd` command, and then listing the contents with the fancy option using the `ls -F` command.

Change directory command

Another way of exploring a directory is to make it the current working directory so that `ls` operates on it by default. You do this with the `cd` (change directory) command (Fig. A.1C.11).

The `cd` command takes a single argument, the directory path, to make the current working directory. The indicated directory then becomes the default directory for `ls` and other file utilities.

Sometimes the shell prompt will indicate the current working directory. By convention, the home directory is indicated using a tilde (~) symbol, so the prompt (`~/docs`) indicates that the current working directory is the `docs` directory inside the home directory.

Print working directory command

If your prompt doesn't have a working directory indicator, you can find out the current directory with the `pwd` (print working directory) command:

```
(~/docs) 72% pwd
  /home/lstein/docs
```

Unlike the shell prompt, `pwd` doesn't indicate the home directory with a tilde (~), but prints out the complete path, which in this case is `/home/lstein/docs`.

Common Commands and Shortcuts

Move/rename command

To move a file or directory from one location to another use the `mv` (move) command. It takes two arguments: the file or directory to move and the location to move it to. For example, the following command will move the directory `networking_tutorial` and all its contents into the directory `talks`:

```
(~/docs) 72% mv networking_tutorial talks
```

The `mv` command can also be used to rename an existing file or directory. This example will rename the file `mod_perl_book.tar.gz` to `modperl_book.tgz`:

```
(~/docs) 73% mv mod_perl_book.tar.gz modperl_book.tgz
```

The difference between the two commands is that in the first case the second argument was an existing directory, and so was interpreted by the `mv` command as an instruction to move the first argument into that directory. In the second case, the second argument was not a directory, and so was interpreted by `mv` as a command to rename the indicated file.

Copy command

The `cp` command will make a copy of a file (but not a directory). It is simple to use:

```
(~) 74% cp INBOX INBOX.bak
```

This creates an identical copy of the file `INBOX` named `INBOX.bak`.

Make and remove directory commands

To create a new directory, use the `mkdir` (make directory) command. This takes a list of one or more directory names and creates them in the current working directory. To remove an empty directory, use `rmdir` (remove directory). This command will fail if the directory is not empty.

Remove command

To delete a file, use the `rm` (remove) command. It takes a list of files and deletes them. The deletion is irrevocable—i.e., unlike Windows and Macintosh systems there is no recycle bin or trashcan from which to retrieve deleted files. A useful variant of `rm` is `rm -r`, which will delete a directory and all its contents; however, be careful with this, as it is easy to delete more than you intend.

Wild cards

The shell provides several convenient shortcuts. One is wild cards which allow you to refer to several files or directories at once. An asterisk "*" appearing in a command line argument is treated as a wild card that can match any series of zero or more characters, while a question mark "?" can match any single character.

```
A
(~) 83% ls -F *.png
 chloroplast.png  plastid.png


B
(~) 84% ls -lF *plastid*
 -rw-r--r--    1 lstein   lstein      26988 Mar  8 06:34 plastid.png
```

Figure A.1C.12 Using wildcards with the `ls` command (fancy option, `-F`) to display all (**A**) PNG files and (**B**) files containing the text "plastid."

```
(~) 86% ls -F ../siao
 4975.pdf  GNUstep/  autosave/  mp3.directory  tranaller.pdf
 Back up/  Mail/     copy/      nsmail/
 Desktop/  News/     mp3/       public_html/
```

Figure A.1C.13 Listing the contents of a directory using the double dot ". ." abbreviation for the root directory.

Using wild cards, you can refer to all PNG files as shown in Figure A.1C.12A, or to all files that contain the text "plastid" as shown in Figure A.1C.12B.

Directory abbreviations

If you are in a nested subdirectory and you want to refer to the directory above the current one, you can refer to this directory with the special name ".." (two dots). For example, the following command, when executed from your home directory, will list the contents of the directory that contains it:

```
(~) 85% ls -F..
  ftp/ lost+found/ lstein/ siao/ testuser/ todd/ www/
```

The ".." can be used in a directory path just like any other directory name as shown in Figure A.1C.13.

The symbol "." stands for the current working directory.

The shell also lets you use the tilde symbol "~" (found in the upper left-hand corner of most keyboards) to refer to your home directory. You can obtain a listing of your home directory like this:

```
(~/docs) 88% ls -F ~
```

and return to your home directory from wherever you are by typing:

```
(~/docs) 89: cd ~
```

For your convenience, typing `cd` alone will also return you to your home directory, making it the current working directory.

WORKING WITH TEXT FILES

Creating and manipulating files of text is central to most bioinformatics activities. Unix gives you a large number of ways to manipulate text files.

```
(~) 104% more projects/data/genomic-seq.fasta
 >HSBA536C5 LOCUS       HSBA536C5  168628 bp    DNA           HTG   16-MAY-2000
 GCTGACGCTGCTGCTGCTGCTAATGCTAGGTAAGCCAGGCCCCCGTTCCGGTTCCCGGCC
 CTGCCGGGCAGGTGCCTGCCATCCAGCACCCCCCTAGCATTTCGGGGCTCCTCACATCCT
 CCTCCGTCAGGGGTACCTCCAGCCTCCCCGGGGTCGGCGTCCAGCTGCGCTTTGTGGCTG
 TCCGCCGAGCCAGTCTTGGGAAGAGTGCAGGCTCCCCAGCTCTAAGCCAGGGACTGGGGC
 ...
 --More--(1%)
```

Figure A.1C.14 Viewing the first 1% of the text file `genomic-seq.fasta`, located in the directory `projects/data`, using the `more` command. Note that some of the screen has been deleted to conserve space.

The fastest and easiest way to view the contents of a text file is with the `more` command. It takes a list of one or more text file names, and displays them on the screen one page at a time. This works even with very large files. See Figure A.1C.14 for example.

The `−More−` prompt at the bottom of the screen indicates that there is more of the file to display and gives the approximate position of the region that is being displayed, in this case the top 1%. As described earlier, you can page through the file from top to bottom by pressing the Space Bar. Press "q" to stop viewing the file.

If your system has it, the `less` command is recommended as an improved version of `more`. It works just like `more`, but allows you to page upwards as well as downwards by pressing the Page Up and Page Down keys. It also allows you to navigate a line at a time using the up (↑) and down (↓) cursor keys, and to search through the file for words and phrases.

Redirecting Output to a File

Much of the software used in bioinformatics produces large amounts of text data. This information is often written directly to the terminal, and it can be frustrating to see something interesting scroll by into the irretrievable oblivion beyond the top of the terminal window.

One way to deal with this situation is to use the output redirection feature of the Unix shell. The output of any command can be redirected into a file by following the command with a "$>$" sign followed by the name of the file you wish to create. For example, the `blastn` command (see *UNIT 3.3*) will write the results of its search to the terminal window. To redirect this to a file named `blastn.out`, issue the command as shown in Figure A.1C.15A. After the command completes, you will find its output in `blastn.out`, which you can then inspect with `more`, `less`, or a text editor.

If the file indicated with "$>$" already exists, it will be overwritten, erasing whatever was there before. If you prefer to append the command output to the file, leaving its previous contents intact, use "$\gg$", as shown in Figure A.1C.15B.

Redirecting Output to More

Another handy alternative, useful for those cases when there is more output from a command than will fit onto a terminal screen, but you don't need to save the information to a file, is to redirect output directly to the `more` program. You can do this using the "pipe" or vertical bar symbol "|":

```
(~) 107% blastn humseq202 data/genomic-seq.fasta | more
```

```
A
(~) 105% blastn humseq202 data/genomic-seq.fasta > blastn.out

B
(~) 106% blastn humseq202 data/genomic-seq.fasta >> blastn.out
```

Figure A.1C.15 Redirecting the results of the `blastn` command to the file `blastn.out`. Note that (**A**) using a single greater than sign "`>`" sign causes any previous copy of `blastn.out` to be overwritten, while (**B**) using a double greater-than sign "≫" will cause the current output to be appended to the existing `blastn.out` file.

Table A.1C.1 Graphical Text Editors

Desktop environment	Editor
Gnome	gedit
KDE	kedit
CDE	dtpad

Blastn's output will now be captured by `more` and displayed a screen at a time for easy viewing.

Unix Text Editors

To work effectively with Unix-based bioinformatics software you will need to be able to create and modify text files from scratch. This means becoming proficient with one or more of the Unix text editors.

Unlike the more familiar word processors, Unix text editors produce files that are devoid of any fancy fonts or formatting. They also have a reputation for being unfriendly to novice users. This is only partly true. The graphical desktop environments are each equipped with user-friendly text editors similar in style to the Windows Notepad desk accessory. Table A.1C.1 lists the names of the graphical text editors in each of the three most popular Unix desktop environments. Each one is easily reachable via a menu item or icon.

If you have access to one of the graphical editors, you should have no problem creating and editing text files since everything can be done using mouse clicks and menu commands. If you must interact with Unix via a text-only terminal, life will be slightly more interesting.

pico

If you have never used a Unix text editor before, it is suggested that you begin with the pico text editor. This editor is installed on most (but not all) Unix systems, and has a relatively straightforward user interface. To launch it, type `pico` at the command line. This will replace the contents of the terminal window with the editor screen shown in Figure A.1C.16. The middle of the screen is the current contents of the text file. Use the cursor keys to move around in the file, the Backspace key to delete text to the left of the insertion point, and the Delete key to delete text to the right of the insertion point.

Various Control key combinations allow you to read files, save files, and exit the program. The currently available commands are listed at the bottom of the pico window using notation in which a caret "^" means the Control key. So `^X Exit` means to press Control-X in order to exit the program.

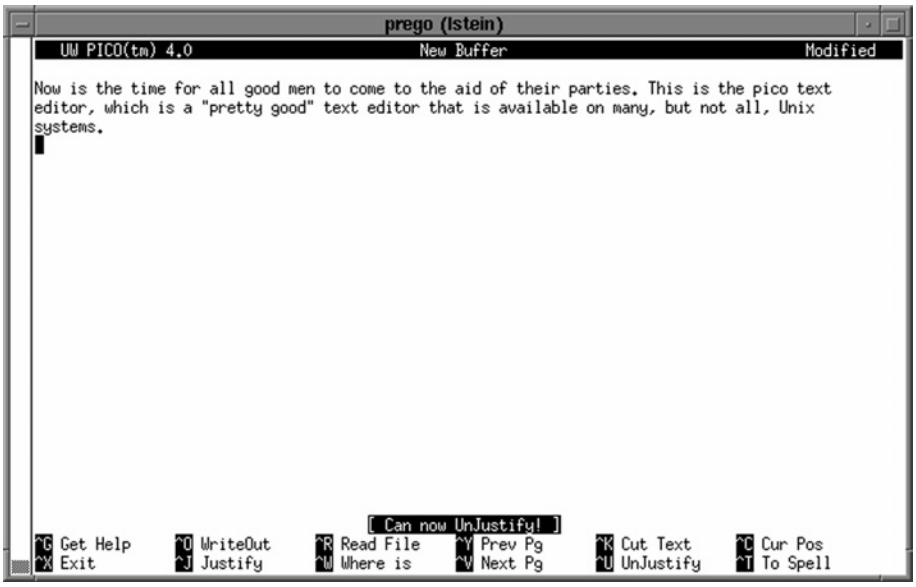

Figure A.1C.16 Pico editor screen.

To create a text file from scratch, launch pico, type the text, and then press Control-O to write out (i.e., output) the file. You will be prompted to type in the name of the file. To read in an existing file, press Control-R. You will be prompted for a file name, which will then be appended to the bottom of whatever is currently on display. Another way to edit an existing file is to give its name to pico on the command line. For example, for the file `test.txt`, the following will cause pico to open and edit the file:

```
(~) 108% pico test.txt
```

Other text editors

Pico has relatively limited abilities. Much more powerful Unix text editors are the aforementioned vi editor, as well as Emacs. Using vi requires learning a set of cryptic keyboard-based commands. Emacs is slightly easier to use in the graphical X Window environment because it provides menus for most common commands, but it is not nearly as straightforward as more familiar word processors. However, if you plan to work heavily in the Unix environment, it is worth investing some time learning one of these editors. Good introductions to vi and Emacs can be found in most general-audience Unix books.

CHANGING THE ENVIRONMENT

Various Unix commands, and several bioinformatics programs, are dependent on "environment variables," a set of configuration variables that are set up for you each time you log into the system. In this section, we walk through a practical example of changing an environment variable.

VISUAL and SHELL variables

Various Unix commands will automatically invoke a text editor for you when needed (e.g., when examining a configuration file). The default text editor is vi, a powerful but extremely cryptic text-based editor. To make pico your default editor, you must alter the value of an environment variable named `VISUAL`.

To change the VISUAL environment variable you must edit one of the hidden "dot" files located in your home directory. Which file you edit depends on which shell interpreter you are using. To discover which shell you are using, you will examine the contents of another environment variable named SHELL. Run the command echo $SHELL:

```
(~) 51% echo $SHELL
  /bin/tcsh
```

The echo command simply echoes back its arguments, which in this case is the contents of the SHELL environment variable. The command will print out the path to the shell program that is currently running, which will most likely be one of /bin/tcsh, /bin/csh, /bin/ksh, /bin/bash, or /bin/sh. If the shell is either tcsh or csh, then the configuration file you will edit is .cshrc. For any other shell, you will instead edit the file .profile.

We will first assume that you are running tcsh or csh, and therefore need to edit the file .cshrc in your home directory. First, make a copy of the current version of .cshrc, using the cp (copy) command. Name the copy cshrc.orig:

```
(~) 52% cp ~/.cshrc ~/cshrc.orig
```

Now using pico, edit .cshrc:

```
(~) 52% pico ~/.cshrc
```

If the file does not already exist, create it. Scroll to the bottom of the file and add this ine:

```
setenv VISUAL pico
```

Save the file, and then log out of the shell. Log in again and confirm that VISUAL is now set to pico:

```
(~) 56% echo $VISUAL
  pico
```

The procedure is slightly different for the bash, ksh, or sh shells. In this case, the file to change is .profile, also located at the top level of your home directory. Create a copy of .profile as described earlier for .cshrc. Using pico (or another text editor), open or create this file, and then add the following two lines:

```
VISUAL=pico
  export VISUAL
```

Log out and in again, and run echo $VISUAL to confirm that the environment variable has indeed been set.

Other variables

You can follow this procedure to add or modify any number of environment variables. Just be sure to put each setenv or export command on a separate line. If you make a mistake, your shell may start misbehaving. Don't panic. Just copy the original back into place:

```
(~) 56% cp cshrc.orig.cshrc
```

INSTALLING SOFTWARE

The last topic that we'll cover in this survival guide is installing and upgrading software. This is a task that is usually best left to a system administrator, but for many readers this is not a viable option.

Most Unix software, bioinformatics included, is distributed in source code form as `tar.gz` files. The `tar` program is first applied to the software to archive its many files and directories into a single file, and then the `gzip` compression program is used to compress the archive for easy transmission over the Internet.

Although there are an infinite number of variations, downloading and installing Unix software follows this general theme:

1. Identify a Web or FTP site that has the desired software and download it to the Unix system.
2. Uncompress and unarchive the package.
3. Read the `README` and/or `INSTALL` documentation.
4. Configure the package.
5. Compile the software.
6. Install the software.

As an ordinary user of the system you can perform all but the very last step of this process; however, the final install of software requires that you have write permission to portions of the Unix system that are usually off-limits to ordinary users. To install software in its usual place requires that you log in as the privileged user known as "root," using the password for the root account; however, if you do not know the root password, you can still install software in your home directory. In the example that follows we will install a new software package as root, and then as an unprivileged user.

The example we will use is the MySQL package, a popular open source relational database that is used as the exemplar of database management systems in Chapter 9.

Downloading (FTP)

The first step is to download the software archive onto the Unix system. You will eventually generate quite a collection of these archives, so create a directory named `src` under your home directory, and make it your current working directory:

```
(~) 101% mkdir src
(~) 102% cd src
```

If the software is located on an FTP site, you will use the `ftp` command to download it. If the software is located on a Web site, you can use Mozilla Firefox (with the `firefox` command) if you have a graphical login, or the text-only Web browser lynx if you are using a terminal emulation program. Both applications are self-explanatory.

In the case of MySQL, we will connect to the FTP site `ftp.mysql.com` using the `ftp` command. When prompted for a username, we enter the name `anonymous`, and give our E-mail address when prompted for a password (Fig. A.1C.17).

```
(~/src) 103% ftp www.mysql.com
 Connected to www.mysql.com.
 220 ProFTPD 1.2.2rc3 Server (MySQL AB) [64.28.67.70]

 Name (www.mysql.com:lstein): anonymous
 331 Anonymous login ok, send your complete email address as your password.

 Password:
 230 Anonymous access granted, restrictions apply.
 Remote system type is UNIX.
 Using binary mode to transfer files.
```

Figure A.1C.17 Login screen of the MySQL FTP site using anonymous as the login name and the user's e-mail address as the password.

A

```
ftp> cd MySQL-3.23
 250 CWD command successful.
```

B

```
ftp> ls
 200 PORT command successful.
 150 Opening ASCII mode data connection for file list.
 -rw-r--r--   1 ftp      ftp       6067955 Sep  9  2001 MySQL-3.23.42-1.i386.rpm
 -rw-r--r--   1 ftp      ftp      11748355 Sep  9  2001 MySQL-3.23.42-1.src.rpm
 ...
 -rw-r--r--   1 ftp      ftp      11814847 Nov 30 12:02 mysql-3.23.46.tar.gz
 -rw-r--r--   1 ftp      ftp      12541083 Dec  9 07:03 mysql-3.23.46a-win-src.zip
 -rw-r--r--   1 ftp      ftp      12596637 Dec  9 07:03 mysql-3.23.46a-win.zip
 ...
 226 Transfer complete.
```

C

```
ftp> get mysql-3.23.46.tar.gz
 local: mysql-3.23.46.tar.gz remote: mysql-3.23.46.tar.gz
 200 PORT command successful.
 150 Opening BINARY mode data connection for mysql-3.23.46.tar.gz (11844905 bytes).
 226 Transfer complete.
 11844905 bytes received in 74.6 secs (1.6e+02 Kbytes/sec)
```

Figure A.1C.18 (**A**) Changing the working directory using the `cd` command, (**B**) listing files using the `ls` command, and (**C**) retrieving the `mysql-3.23.46.tar.gz` file using the `get` command, all within the FTP program shell. We chose the `mysql-3.23.46.tar.gz` file after determining that is was the most recent version of the MySQL distribution. Note that some of the listing has been omitted to conserve space.

Get command

After logging into the remote site, the prompt changes to `ftp>`, indicating that the command line is now under control of the FTP program. The FTP program contains a miniature shell that recognizes the Unix `cd` and `ls` commands, with the difference that these commands operate on the remote FTP site rather than on the local machine. Using these commands we navigate to the desired directory and use the `get` command to download the file containing the MySQL source distribution (Fig. A.1C.18). We choose the `.tar.gz` file with the most recent date. Other files in this subdirectory end with the Microsoft Windows extension `.zip`.

Quit command

When the download is finished, we issue the `quit` command, and the FTP program exits, returning us to the Unix shell:

```
ftp> quit
221 Goodbye.
(~/src) 104%
```

Uncompressing and unarchiving

The next step is to unpack the MySQL distribution. Return to the home directory and create a new directory named `build`. This will be used as a temporary place in which to build new software prior to installing it:

```
(~/src) 105% cd ~
(~) 106% mkdir build
(~) 107% cd build
```

We will now uncompress and unarchive the MySQL distribution in a single step. This uses a trick in which the output of the gunzip program, which uncompresses the archive, is fed directly into the input of the tar program, which unarchives the software. The magical incantation and its results are shown in Figure A.1C.19. Notice how the "~" symbol is used as a shortcut to indicate the home directory. This will create a directory named `mysql-3.23.46` containing the unpacked MySQL source code distribution. As each file is unpacked, its name is printed on the terminal.

Reading Documentation

We enter this new directory and look for a file named `README`, `INSTALL`, or something similar. In this case there is a `README` file, which contains a general description of MySQL, and a more specific file named `INSTALL-SOURCE` which contains step-by-step instructions on building and installing the software.

Configure Package

MySQL is typical of software that is written in the C programming language. You first run a script contained within the distribution directory called `configure`. This checks that any libraries or other software on which the package depends are installed, and configures the package with values that are appropriate for the variant of Unix that the machine is running. After configure successfully completes, run the `make` program, which compiles the source code into machine-readable computer code. Finally, you give the command `make install` to move the compiled code into the appropriate locations for installed software.

```
(~/build) 108% gunzip -c ~/src/mysql-3.23.46.tar.gz | tar xvf -
 mysql-3.23.46/
 mysql-3.23.46/Makefile.in
 mysql-3.23.46/README
 mysql-3.23.46/stamp-h.in
 mysql-3.23.46/Makefile.am
 mysql-3.23.46/acconfig.h
 mysql-3.23.46/acinclude.m4
 mysql-3.23.46/aclocal.m4
 ...
```

Figure A.1C.19 Uncompressing and unarchiving the MySQL distribution in a single step. Note that the file listing, which runs to hundreds of lines, has been truncated in the interest of space.

```
(~/build/mysql-3.23.46) 120% ./configure
 creating cache ./config.cache
 checking host system type... i686-pc-linux-gnu
 checking target system type... i686-pc-linux-gnu
 checking build system type... i686-pc-linux-gnu
 ...
 Remember to check the platform specific part in the reference manual for
 hints about installing on your platfrom. See the Docs directory.

 Thank you for choosing MySQL!
```

Figure A.1C.20 Invocation and results of the `configure` script in the `mysql-3.23.46` directory. Note that the middle portion of the output has been omitted.

```
(~/build/mysql-3.23.46) 121% make
 cd include; make link_sources
 make[1]: Entering directory '/usr/local/home/lstein/build/mysql-3.23.46/include'
 /bin/cp ../config.h my_config.h
 make[1]: Leaving directory '/usr/local/home/lstein/build/mysql-3.23.46/include'
 echo timestamp > linked_include_sources
 cd libmysql; make link_sources
 ...
 make[2]: Nothing to be done for 'all-am'.
 make[2]: Leaving directory '/usr/local/home/lstein/build/mysql-3.23.46'
 make[1]: Leaving directory
'/usr/local/home/lstein/build/mysql-3.23.46'
(~/build/mysql-3.23.46) 122%
```

Figure A.1C.21 Running the `make` command. Note that most of the output has been omitted.

Configure script

We will step through this process. First, we run the `configure` script located in the `mysql-3.23.46` directory. Since there might be other configure programs installed on your machine, we take care to run the particular one that MySQL comes with by using a path starting with "." to indicate the configure script located in the current working directory as shown in Figure A.1C.20.

Thankfully the configure script ran to completion. If it had detected that some software package that MySQL depends on was missing from the system, it would have failed part way through and notified us of the problem.

Compile

Make program

We now run `make`. The make program is a standard part of Unix rather than a MySQL-specific script, so we do not need to specify the current directory in its path (Fig. A.1C.21). Making the package is an involved process that takes several minutes to complete. If any errors are encountered during the process, make will terminate with any obvious `**error**` message (you can safely ignore any warnings that are issued). If make does fail, your best option is to refer the problem to someone more knowledgeable. Otherwise, you can proceed to `make test` and `make install` (see below).

Make test

Some software packages come with a set of tests that you can run to ensure that they have compiled correctly. If such tests are defined, you can invoke them with the command `make test` as shown in Figure A.1C.22.

```
(~/build/mysql-3.23.46) 122% make test
 cd mysql-test ; ./mysql-test-run
 Installing Test Databases
 ...
  TEST                          USER   SYSTEM  ELAPSED        RESULT
 ---------------------------------------------------------------------
 alias                          0.09     0.06     0.26       [ pass ]
 alter_table                    0.02     0.03     0.14       [ pass ]
 analyse                        0.03     0.02     0.06       [ pass ]
 ...
 variables                      0.04     0.00     0.08       [ pass ]
 warnings                       0.04     0.01     0.06       [ pass ]
 ---------------------------------------------------------------------
 ...
 All 131 tests were successful.
 (~/build/mysql-3.23.46) 123%
```

Figure A.1C.22 Using the software-included `make test` command. Note that most of the output has been omitted for brevity.

All tests passed, so we can feel confident that MySQL will function properly. For packages that do not have any tests defined, the `make test` command will produce an error message similar to `Don't know how to make test. Stop.` In this case just skip the step.

Install

Su command

The last step is to `make install`. The catch here is that you will have to log in as the root user in order to run this command. Assuming that you know the root password, you can use the `su` command to temporarily assume the identity of root without having to log out and in again:

```
(~/build/mysql-3.23.46) 124% su
  Password: ********
  bash-2.05#
```

After issuing the `su` command, the system prompts us for the root password. After entering it, the shell prompt changes, telling us that we are now logged in as the root user (the prompt character "#" is usually, but not necessarily, reserved for root). You can skip this step if the Sudo program is installed and configured, as described later.

Make install command

We then run `make install`, and wait while the system copies the MySQL software into its installed locations as shown in Figure A.1C.23.

Exit command

After `make install` completes, we issue the `exit` command to return to our normal user privileges.

```
bash-2.05# exit
(~/build/mysql-3.23.46) 125%
```

You do not want to remain logged in as root longer than you need to, because as root you have access to commands that can seriously damage the system if issued inadvertently.

```
bash-2.05# make install
 Making install in include
 make[1]: Entering directory '/usr/local/home/lstein/build/mysql-3.23.49/include'
 make[2]: Entering directory '/usr/local/home/lstein/build/mysql-3.23.49/include'
 make[2]: Nothing to be done for 'install-exec-am'.
 /bin/sh ../mkinstalldirs /usr/local/include/mysql
 /usr/bin/ginstall -c -m 644 dbug.h /usr/local/include/mysql/dbug.h
 /usr/bin/ginstall -c -m 644 m_string.h /usr/local/include/mysql/m_string.h
 /usr/bin/ginstall -c -m 644 my_sys.h /usr/local/include/mysql/my_sys.h
 ...
```

Figure A.1C.23 Copying the MySQL software into its installed locations using the `make install` command. Note that most of the output has been omitted for brevity.

There are now several steps that are specific to MySQL, including setting up databases and user accounts. These steps are described in the `INSTALL-SOURCE` file. Since they are not applicable to the general case, we won't cover them here.

Make install with Sudo

On many shared Unix systems you will not have access to the root password. On such systems, the system administrator may be able to give you limited root privileges using the Sudo system. On such systems you may be able to skip the `su` step entirely and run the privileged install step by issuing the command `sudo make install`. The system will prompt you for your password, become the root user temporarily, run `make install`, and then return you to your normal user status. Please see your system administrator for help with this. If you are the system administrator, you can read about how to configure Sudo by consulting the `sudo` and `visudo` manual pages (see The Manual Command, above).

Installing Software into your Home Directory

What if you don't have the root password or Sudo privileges? With a little additional effort, you can install the software package in your home directory, something that you don't need root access to perform.

The key is to pass the optional `--prefix=` option to the configure script, specifying a location in your home directory after the equal sign. My home directory is `/home/lstein` and I would like MySQL to install itself in a subdirectory named `mysql`, so I pass the option `--prefix=/home/lstein/mysql`, as shown in Figure A.1C.24.

Be sure to use the full path to your home directory here. If you are unsure of the correct value, `cd` to your home directory and then issue the `pwd` command.

Now run the `make` and `make test` commands as described earlier. If all goes well, run `make install`. Since you are installing into your home directory, there is no reason to become root.

MySQL has been installed in your home directory. What now? If you inspect the contents of the `~/mysql` directory, you will discover that the installation process created a number of subdirectories (see Managing Files and Directories; Fig. A.1C.25).

By convention the `bin` subdirectory contains executable files (commands), `man` contains documentation, and `include` and `lib` together contain packages of code for use by software developers. The other directories contain mysql-specific components. The `mysql` program lives in `~/mysql/bin`, and you can run it by typing its complete path:

```
(~/build/mysql-3.23.46) 135% ./configure --prefix=/home/lstein/mysql
```

Figure A.1C.24 Providing a directory into which MySQL can install itself.

```
(~/build/mysql-3.23.46) 140% ls ~/mysql/
 bin/      info/  libexec/   mysql-test/   sql-bench/
 include/  lib/    man/      share/        var/
```

Figure A.1C.25 Subdirectories of ~ created during installation.

```
(~/build/mysql-3.23.46) 141% ~/mysql/bin/mysql
```

If you like, you can set up your environment so that `~/mysql/bin` is searched automatically whenever you type a command. This involves setting the environment variable `PATH`, which contains a list of directories to be searched for executables (see List Command).

As described earlier, the procedure to follow depends on which shell you are using. If you are using `tcsh` or `csh`, open the file `.cshrc` and add the following to the bottom:

```
setenv PATH ~/mysql/bin:$PATH
```

This will set the `PATH` environment variable to contain `~/mysql/bin`, followed by whatever was on the `PATH` before. Log out and in again. You should now be able type `mysql` without qualifying it with a path.

If you are using the `bash`, `ksh`, or `sh` shells, open `.profile` and add the following to the bottom:

```
export PATH=~/mysql/bin:$PATH
```

Again check that when you log out and in again, `mysql` is found automatically.

As before, you are advised to make copies of `.cshrc` or `.profile` before you do this. If you mess up `PATH`, the system may not be able to find any commands, including the `cp` command required to restore the original version of `.cshrc` or `.profile`. This isn't a cataclysm. Simply refer to `cp` using its explicit path, `/bin/cp`:

```
(~) 142% /bin/cp cshrc.orig.cshrc
```

This will put `.cshrc` back the way it was before you modified it.

If you install a program and later move it to another location on your `PATH`, the system may not be able to find it until you log off and in again. With the `csh` and `tchs` shells, the command `rehash` may help the system find the command without doing this.

CONCLUSION

Unix will feel alien and intimidating at first. Do not be inhibited, but feel free to explore and experiment with the Unix environment. With experience, you may eventually come to tolerate, if not appreciate, Unix's alternative take on the world.

KEY REFERENCES

Frisch, A. 1996. Essential System Administration, 2nd Edition. O'Reilly and Associates, Sebastopol, Calif.
A slightly more advanced book that emphasizes troubleshooting.

Nemeth, E., Snyder, G., and Seebass, S. 1995. Unix System Administration Handbook. Prentice-Hall, Engelwood Cliffs, NJ.
This is a user friendly and comprehensive guide to working on Unix systems. Although aimed at system administrators, it is highly recommended for newcomers to the Unix environment.

Raymond, E.S. (ed.) 1996. The New Hacker's Dictionary, Third Edition. MIT Press, Cambridge, Mass.
An introduction to the Unix culture.

Sobel, M.G. 1998. Hands-On Linux. Addison-Wesley, Reading, Mass.
An introduction to Linux.

Welsh, M. 1999. Tunning Linux, Third Edition. O'Reilly and Associates, Sebastopol, Calif.
A user-level guide to the Linux operating system

INTERNET RESOURCES

Login Packages for Windows

http://www.microsoft.com
Microsoft Web site for downloading Telnet. Bare-bones terminal emulator using the Telnet protocol and text-only login. Built into Windows 95 & higher.

http://www.securenetterm.com
NetTerm Web site. More configurable terminal emulator using Telnet and Secure Shell protocols using text-only Web site.

http://www.vandyke.com/products/crt
CRT Web site. Full-featured terminal emulator using Telnet and rlogin protocols using text-only login.

http://www.chiark.greenend.org.uk/~sgtatham/putty/
PuTTY Web site. Bare-bones terminal emulator using Telnet and Secure Shell protocols using text-only login (freeware).

http://www.uk.research.att.com/vnc
VNCviewer Web site. Graphical login using the lightweight VNC protocol (freeware).

http://www.hummingbird.com/products/nc/exceed/
eXceed Web site. Graphical login using the network- intensive X Windows protocol.

http://www.powerlan-usa.com
WebTermX Web site. Graphical login using the network- intensive X Windows protocol

http://www.starnet.com/products
X-Win32 Web site. Graphical login using the network- intensive X Windows protocol.

Contributed by Lincoln D. Stein
Cold Spring Harbor Laboratory
Cold Spring Harbor, New York